



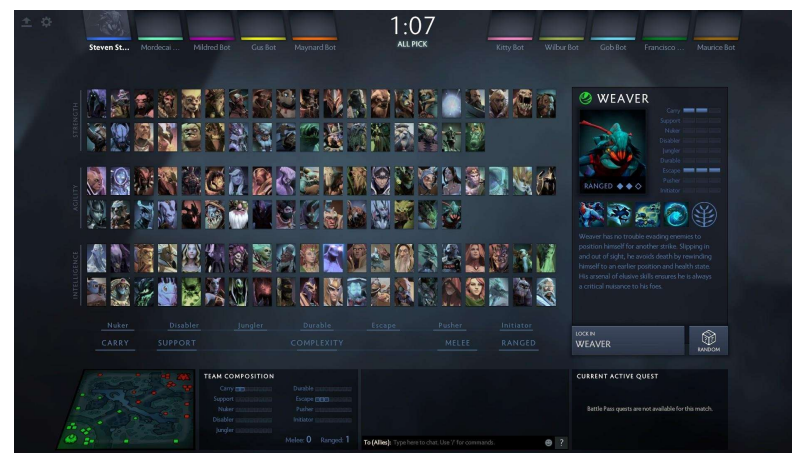
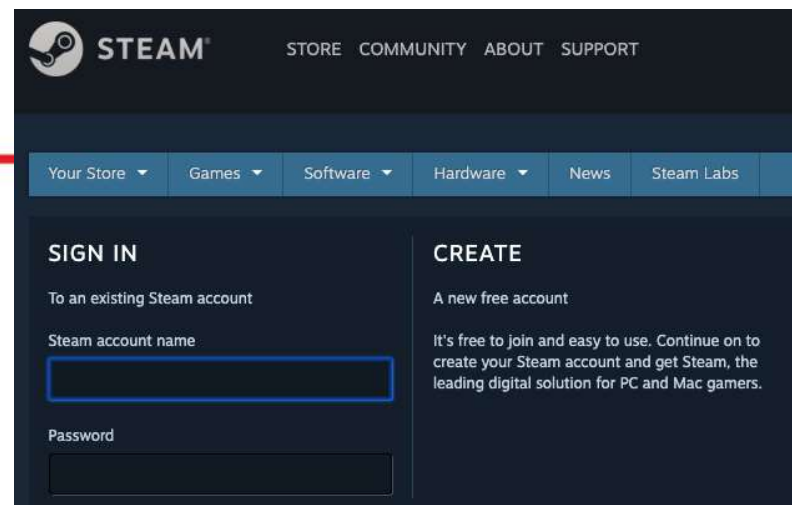
Distance Learning System

# TCP i UDP protokoli

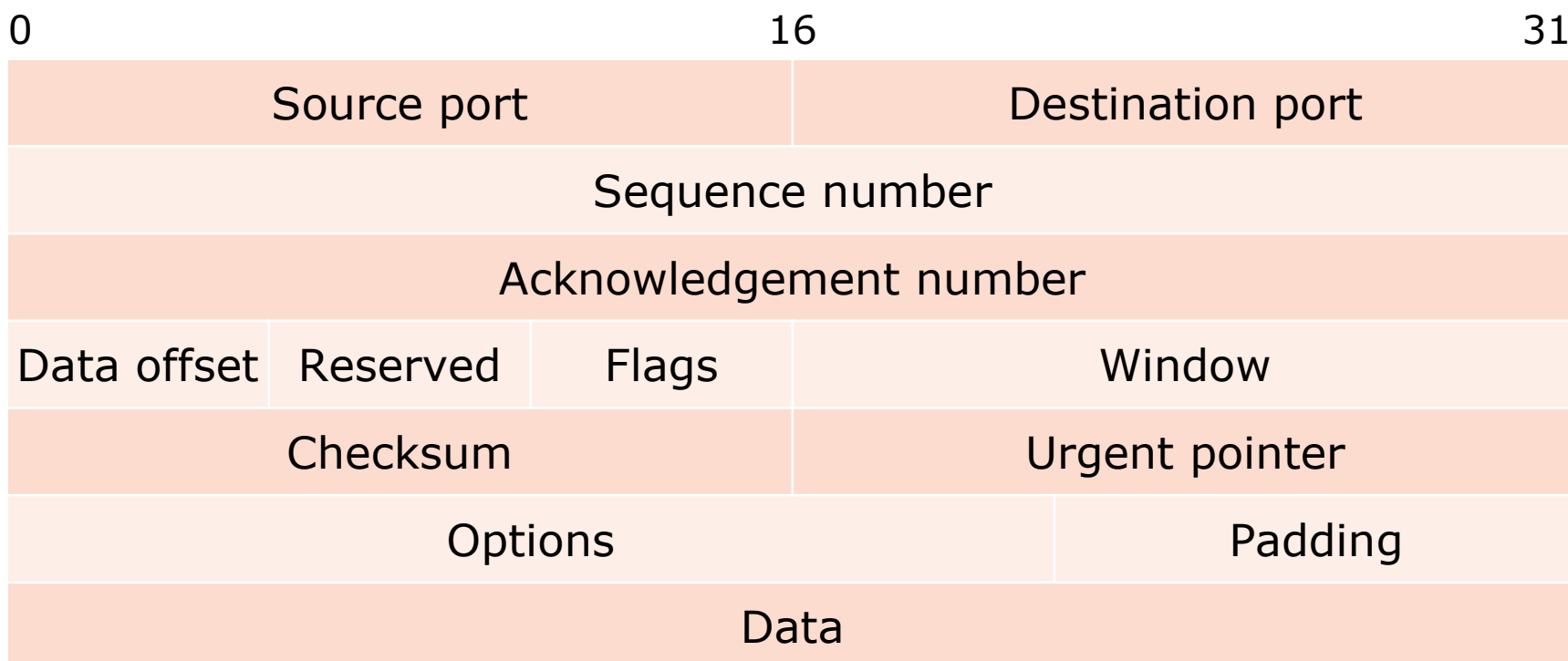
Python Network Programming

# TCP Protokol

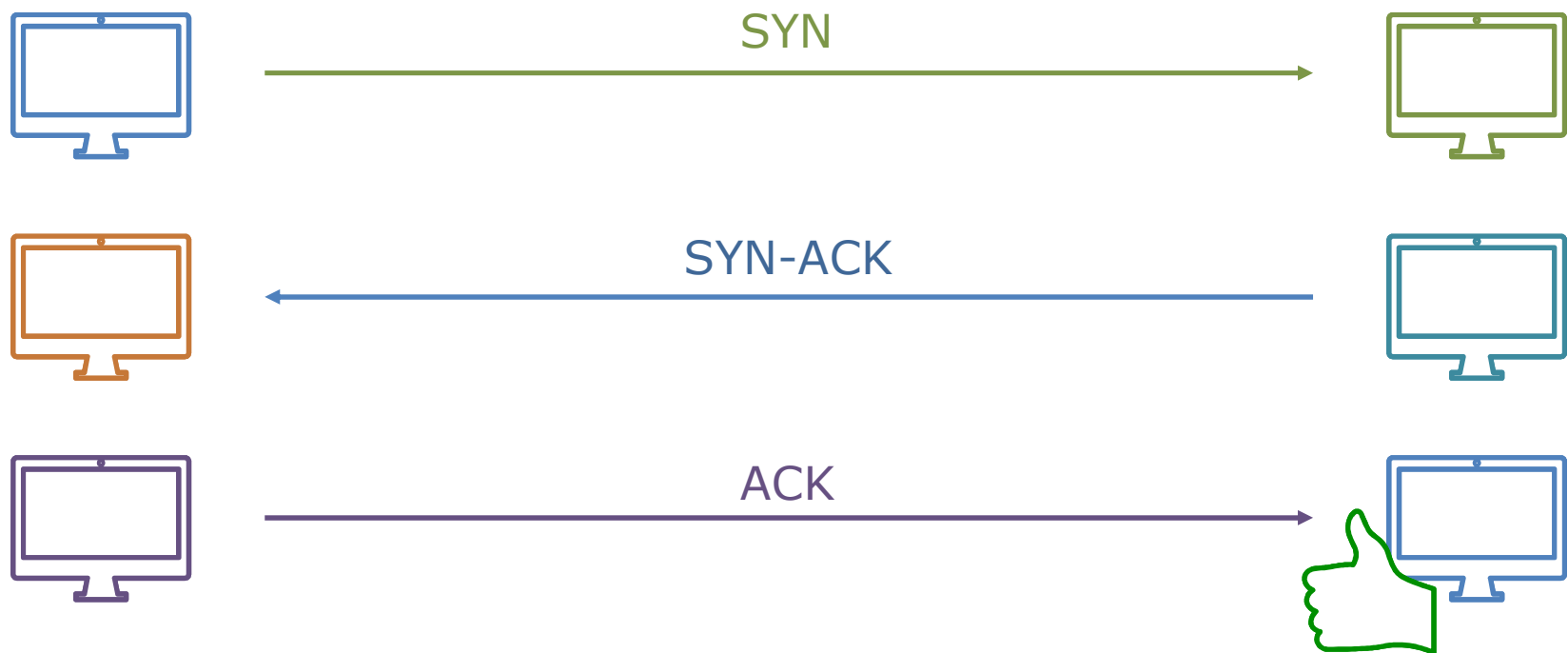
- Kada se ostvari konekcija između pošiljaoca i primaoca u TCP protokolu, ona biva održavana sve do trenutka dok paket ne bude prihvaćen. To omogućava dvosmernu komunikaciju između korespondenata (računara)
- Njegove karakteristike su:
  - Visok nivo pouzdanosti i zato njega srećemo u većini internet protokola višeg nivoa (FTP, HTTP, SMTP...).
  - Slabija brzina rekonektovanja (nije pogodan za aplikacije koje zahtevaju veliku brzinu komunikacije (realno vreme))



# Paket TCP protokola

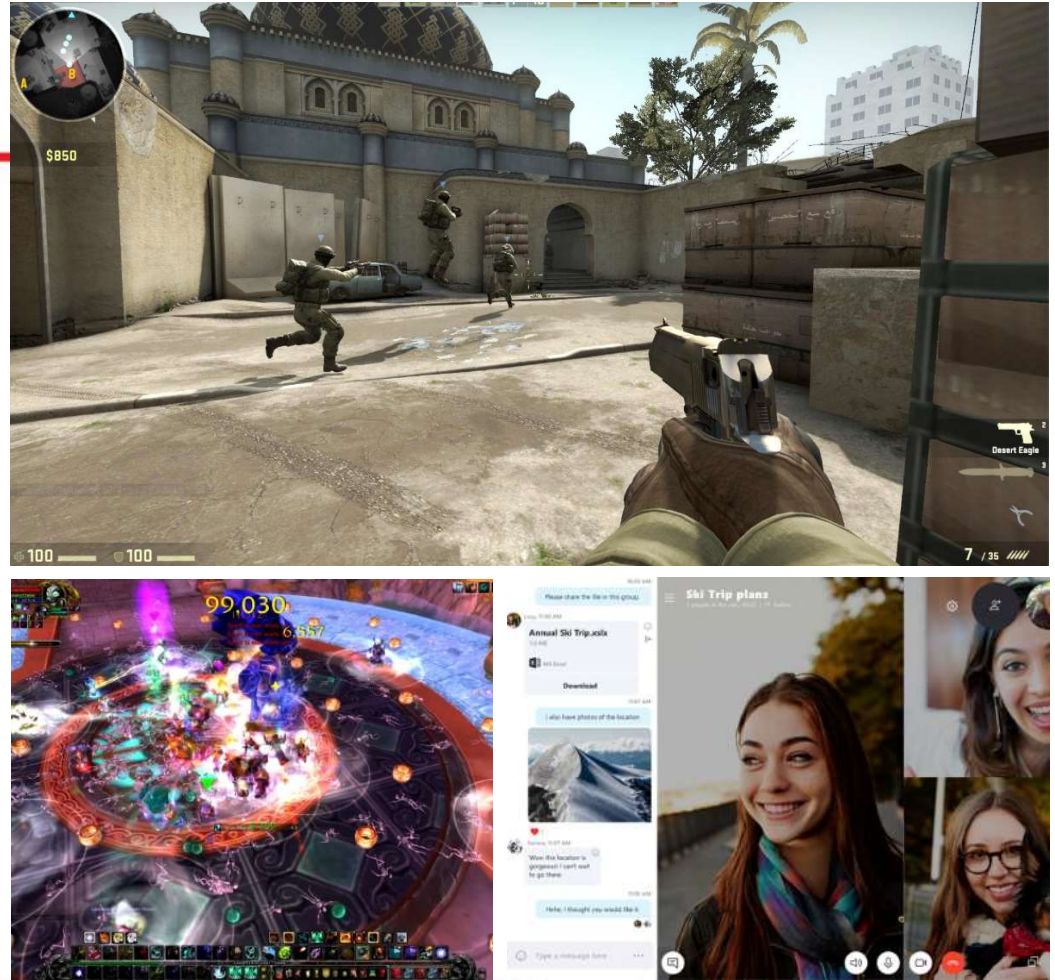


# TCP konekcija (handshake)



# UDP Protokol

- UDP ne zahteva perzistentnu konekciju između krajnjih tačaka komunikacije
- Njegove karakteristike su:
  - Brzina i efikasnost
  - Slab integritet podataka



TCP i UDP protokoli

**LINKgroup**

# Paket UDP protokola

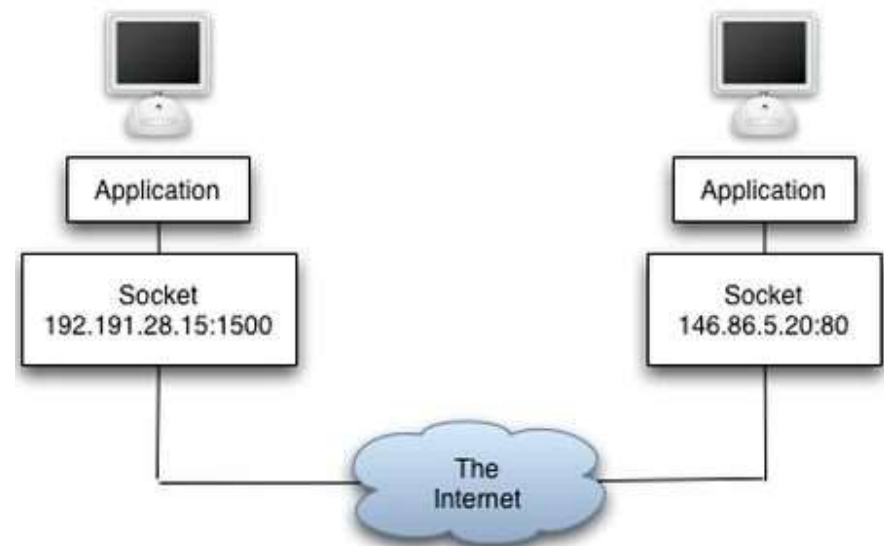
---

0	16	31
Source port		Destination port
Length		Checksum

# Socket

(<http://beej.us/guide/bgnet/>)

- Socket predstavlja jedan kraj u komunikaciji između dva procesa. Određen je IP adresom, koja identifikuje host, i brojem porta koji identifikuje proces koji se izvršava na mrežnom čvoru
- Postoji nekoliko tipova Socketa u zavisnosti od tipa konekcije:
  - **Datagram socket**, socketi koji ne zahtevaju otvorenu konekciju, koriste se sa UDP protokolom
  - **Stream socket**, socketi za rad u konektovanom okruženju sa TCP protokolima
  - **Raw socket**, uglavnom implementiran u ruterima i drugoj mrežnoj opremi



# TCP Socket programiranje

---

- Socket predstavlja IP adresu i port jednog sagovornika u konekciji. On počinje da postoji onog trenutka kada se konekcija ostvari i prekida svoje postojanje nakon što se konekcija prekine. Pri tome je, kod TCP protokola, ova konekcija je **dvosmerna** i **asinhrona**.
- Da bi postojala neka konekcija, potrebne su dve strane: **server** i **klijent**.
  - **Server** ima ulogu slušaoca. On sluša sve aktivnosti na zadatom portu. Nakon što se konekcija dogodi, server inicijalizuje Socket i izvršava definisane aktivnosti. Dakle, pasivan je sve dok klijent ne inicira promenu njegovog statusa.
  - **Klijent** je taj koji inicira komunikaciju sa serverom. Nakon što prosledi svoj zahtev, on očekuje od servera odgovor čime se uspostavlja uzajamna razmena podataka. Sve dok je klijent „nakačen“ na server, razmena podataka traje.
- Razmena podataka biva prekinuta kada klijent prekine komunikaciju sa serverom.
- Python ima više sistema za rad sa soketima, na različitim nivoima (od sirovih soketa, do kompletne HTTP podrške)
- U osnovi rada sa soketima u Python-u, nalazi se paket **socket**. Biblioteka koja radi po principima BSD socket API-ja



# BSD API

---

socket	→	Kreiranje soketa
bind	→	"Zauzimanje" porta
listen	→	Slušanje
accept	→	Prihvatanje dolazne konekcije
connect	→	Ostvarivanje odlazne konekcije
send	→	Slanje podataka
recv	→	Preuzimanje podataka
close	→	Zatvaranje soketa

# Kreiranje TCP slušača (pnp-ex01 socketlisten.py)

socket.AF\_UNIX  
**socket.AF\_INET**  
socket.AF\_INET6

**AF** = Address Family

**socket.SOCK\_STREAM**  
socket.SOCK\_DGRAM  
socket.SOCK\_RAW  
socket.SOCK\_RDM  
socket.SOCK\_SEQPACKET

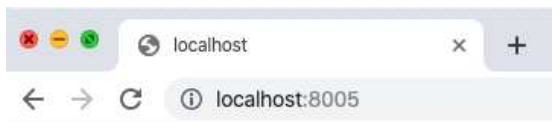
```
import socket

sSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sSocket.bind(("127.0.0.1", 8005))
sSocket.listen()
print("Server is listening...")
sSocket.accept()
```

Kritična operacija jer  
zahteva rezervaciju porta

Blokirajuća operacija

Program će prekinuti izvršavanje nakon prve ostvarene TCP konekcije:



```
# telnet 127.0.0.1 8005
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Connection closed by foreign host.
```

# Kreiranje TCP konekcije (pnp-ex01 socketclient.py)

---

```
import socket

sClient = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sClient.connect(("127.0.0.1", 8005))
```

Server postoji

```
# python3 socketclient.py
#
```

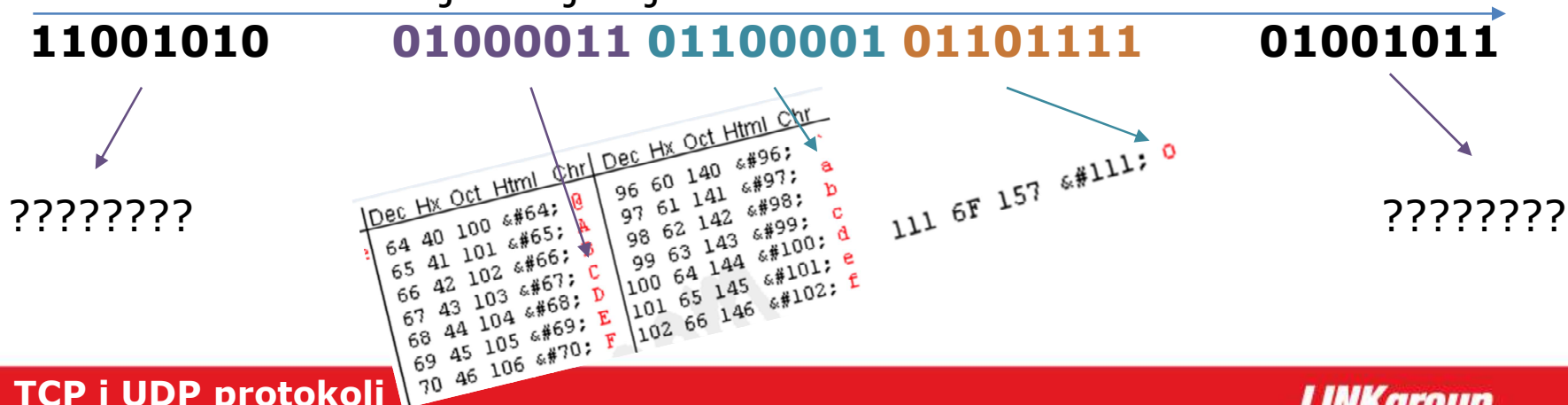
Server ne postoji

```
# python3 socketclient.py
Traceback (most recent call last):
  File "socketclient.py", line 4, in <module>
    sClient.connect(("127.0.0.1", 8005))
ConnectionRefusedError: [Errno 61] Connection refused
```

# Razmena podataka

- TCP konekcija je strimovana
- Strimovana komunikacija podrazumeva tok bajtova u jednom ili dva pravca
- Prilikom čitanja toka bajtova, uvek se postavljaju minimum dva pitanja

1. Sta bajtovi predstavljaju?
2. Gde je kraj bajtova?



# Čitanje podataka iz TCP toka (pnp-ex01 sockrecv.py)

```
import socket
import time

sSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM);
sSocket.bind(("localhost",8005))
sSocket.listen()
print("Listening for connection: ")
conn, addr = sSocket.accept()

print("Connection received from ",addr)
data = conn.recv(1024)

print("Message from client: \n")
time.sleep(2)
print(data)
conn.close()
sSocket.close()
```

- Za čitanje dolaznih podataka (kao i za slanje podataka) koristi se konekcionni objekat (**socket**) dobijen od accept metode
- Metod **recv** dobijenog objekta omogućava čitanje dolaznih podataka
- Metod prihvata parametar - broj bajtova koje treba pročitati
- Metod recv kao rezultat vraća niz bajtova (**bytes** objekat)

## Upis podataka u TCP tok (pnp-ex01 socksend.py)

- Bajtovi se mogu poslati u tok, metodama **send** i **sendall** (takođe postoje i druge metode za slanje podataka u tok)
- Metod **send**, šalje bajtove i vraća broj uspešno poslatih bajtova
- Metod **sendall** šalje bajtove, i nema povratnu vrednost, ali garantuje slanje ukoliko je konekcija ostvarena

```
import socket

cSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
cSocket.connect(("localhost", 8005))

cSocket.send(b"Hey man, what's up?")
cSocket.sendall(b"Are you there?")

cSocket.close()
```

Šalje sve bajtove u tok

# Sinhronizacija između "sagovornika" u TCP

- Da bi sagovornici mogli da komuniciraju, moraju utvrditi pravila tog "razgovora". Ova pravila, u mrežnom programiranju, nazivaju se **protokoli**
- Protokol koji ćemo koristiti u sopstvenoj aplikaciji, može biti utvrđen po nekom postojećem standardu, biti u potpunosti autorski, ili kombinacija jednog i drugog
- Najčešći problem u strimovanoj mrežnoj komunikaciji jeste detekcija završetka poruke



*Bas mi se sviđaš ali mi se uopšte ne sviđaš*


*Samo ti pričaj slušam te ja*



# Izolovanje poruka pomoću delimitera

- Jedan od načina da se izoluju poruke jeste korišćenje delimitera
- Delimiter treba da bude oznaka čije pojavljivanje ne postoji u sadržaju poruke


Delimiter je #



Everything after  
# is bullshit

*Bas mi se svidjaš#ali mi se uopšte ne sviđaš*

*Ajde sevaj#Šalim se, nisam tako mislila*

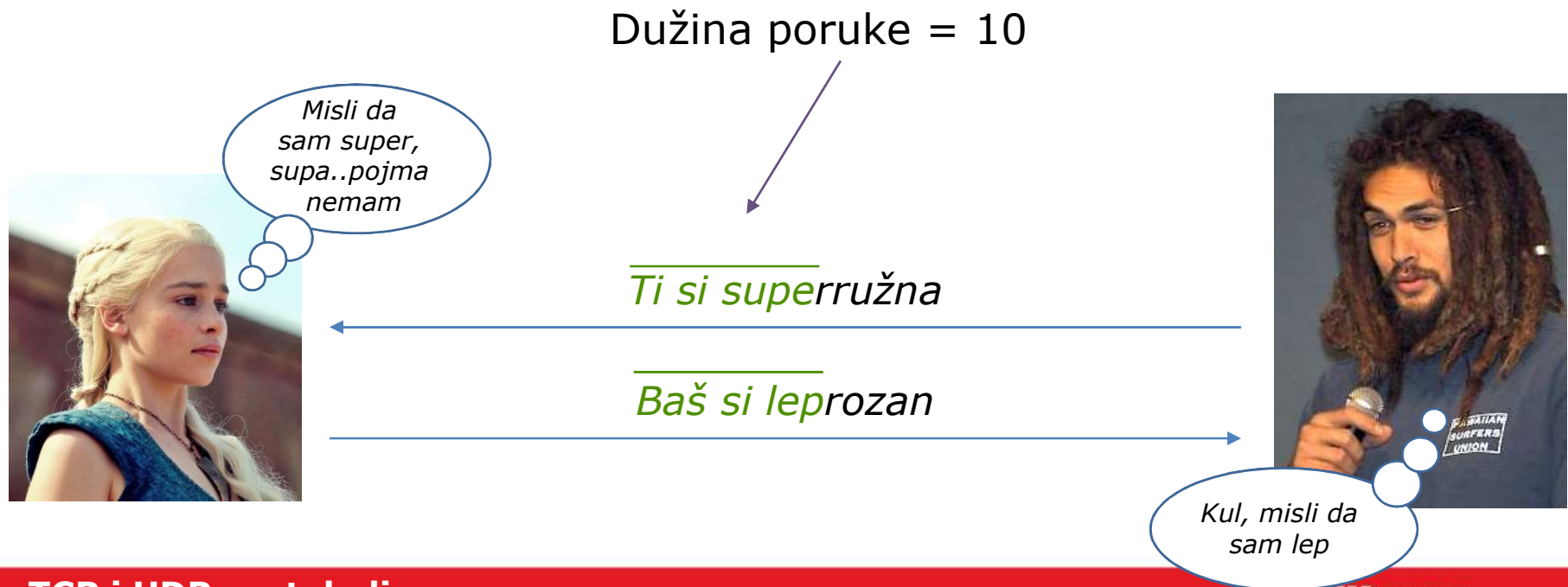


Čitam samo  
do delimitera



# Izolovanje poruka pomoću dužine

- Poruke se takođe mogu izolovati definisanjem dužine poruke
- Tada moramo paziti da sadržaji odgovaraju definisanoj dužini



# Sinhronizacija tipa sadržaja

- Osim dužine poruka, problem u strimovanoj komunikaciji je i tip sadržaja
- Poruke mogu imati različita tumačenja od strane onoga ko ih piše i onoga ko ih čita
- Sagovornici bi trebalo da koriste isti sistem za interpretaciju razmenjenih bajtova poruka

Python žena



Sta ovaj  
prica?

$\$x = 100$

*answer = False*

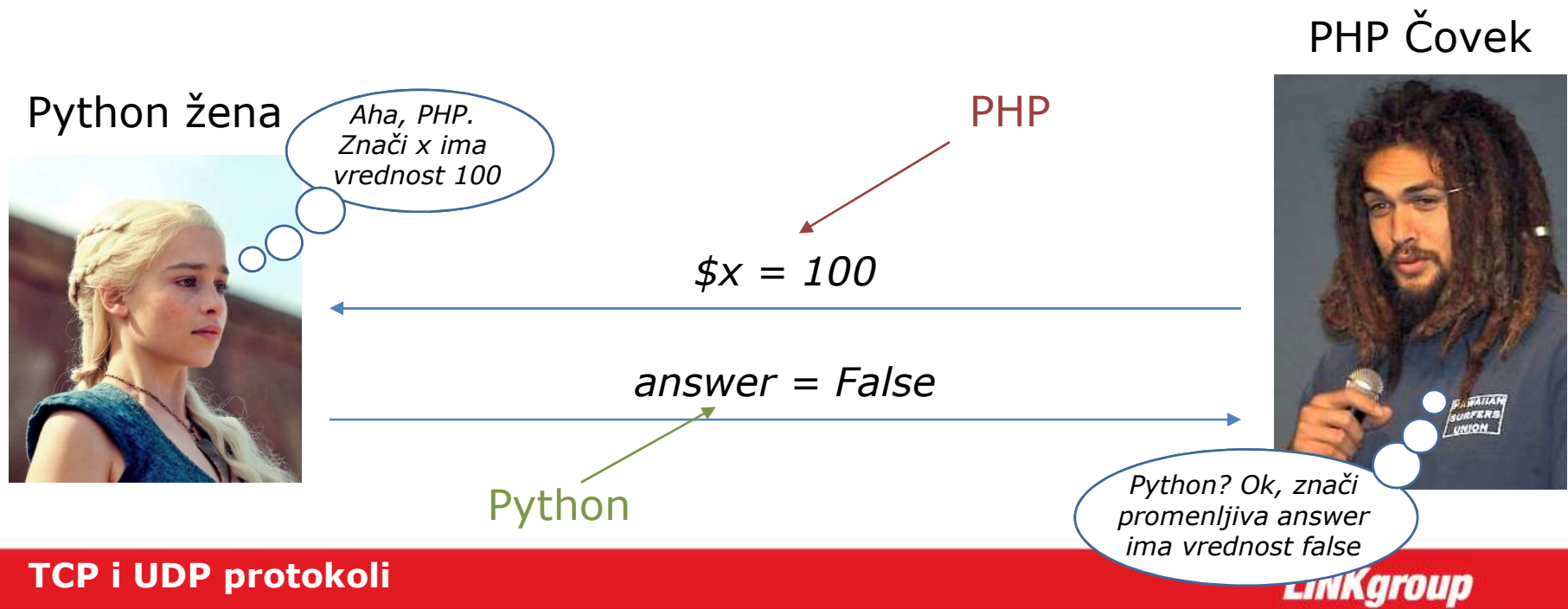
PHP Čovek



Syntax  
error!

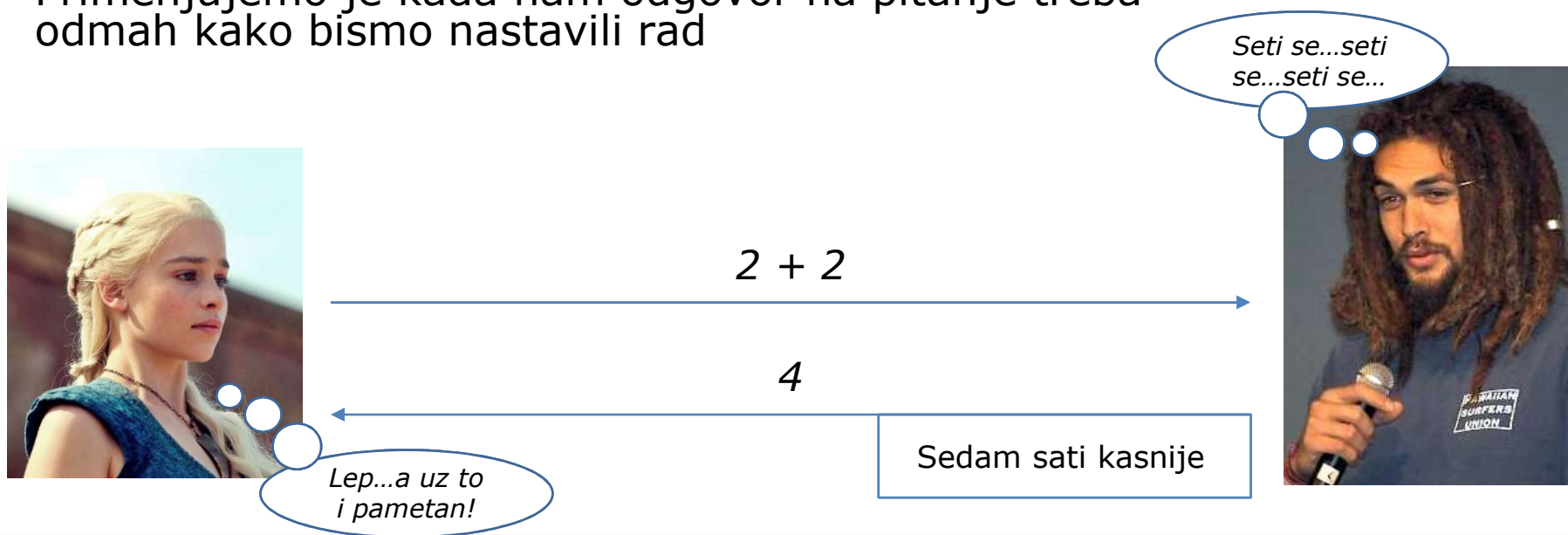
# Sinhronizacija tipa sadržaja

- Ukoliko je oba sagovornika tačno znaju koju vrstu sadržaja sadrži poruka, ne postoji problem u komunikaciji



# Sinhronizacija slanja i primanja sadržaja

- U mrežnoj komunikaciji, mora se znati kada jedan sagovornik "govori", a kada drugi
- Ovo je **blokirana / sinhrona** komunikacija
- Primenujemo je kada nam odgovor na pitanje treba odmah kako bismo nastavili rad



# Asinhrono slanje i primanje sadržaja

- Ponekad, sagovornici mogu "govoriti" i istovremeno ali tada se obično ne očekuje momentalni odgovor već radije samo jedan sagovornik obaveštava drugog o nečemu
- Ovakav sistem se obično realizuje **asinhrono**

Dal' su  
deca ručala?



Hej, zmaj!

Danas sam bila na pijaci da kupim paprike. Posle sam išla kod manikira i pedikira, pa kod frizera. Prefarabala sam se iz plavo plavog, u plavo plavo plavkasto, plavušavo. Kasnije ću da gledam umetničko klizanje, pa ću onda da odgledam novu epizodu serije "Breaking bad". Ima na tv-u, posle **utakmice**. Što me iznervirala koleginica na poslu, zamisli, rekla mi...

Vidiš, ovo ti je avion na daljinsko upravljanje. Ovde pomeraš ako hoćeš da letiš levo i desno, a ovde za gore dole. Avion leti kao **zmaj**. Na ekranu je ispisano na kojoj si visini, pa ako hoćeš da sletiš...

Dal da se kladim  
iz keca u iks?



Hej, utakmica!

# Implementacija sinhronizacije (pnp-ex01 sync)

- Sinhronizacija poruka zavisi od protokola
- Recimo da protokol podrazumeva sledeće korake
  - 1. Konektovanje
  - 2. Server kaže nešto klijentu
  - 3. Klijent odštampa šta je server rekao i kaže nešto serveru
  - 4. Server odštampa šta je klijent rekao, odgovara klijentu porukom zahvalnosti i komunikacija se prekida

server.py

```
import socket
sSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sSocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
sSocket.bind(("localhost", 8005))
sSocket.listen()
conn, addr = sSocket.accept()
2| conn.send(b"Hello and welcome")
4| print("Client said: ", conn.recv(256).decode("utf-8"))
   conn.send(b"Thank you")
   sSocket.close()
```

client.py

```
import socket
sClient = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
1| sClient.connect(("localhost", 8005))
   print("Server said: ", sClient.recv(256).decode("utf-8"))
3| msg = input("msg: ")
   sClient.send(bytes(msg, "utf-8"))
   print("Server said: ", sClient.recv(256).decode("utf-8"))
   sClient.close()
```

## Vežba 1 (pnp-ex01 simplecalc)

---

- Napraviti mrežni program za sabiranje brojeva
- Klijent unosi dva broja, a server vraća rezultat operacije

```
Enter number 1: 125  
Enter number 2: 351  
Result is: 476
```

## Vežba 2 (pnp-ex01 guessnumber)

---

- Server bira broj od 1 do 5
- Klijent unosi broj u konzoli
- Broj se šalje na server
- Server odgovara da li je broj pogođen ili ne

```
Enter number: 4  
Wrong! I guessed 1 and you typed 4
```

```
Enter number: 3  
Nice! I guessed that number
```



## Vežba 3 (pnp-ex01 paperstonescissors)

---

- Napraviti mrežni program papir, kamen, makaze
- Server i klijent biraju predmet
- Klijent šalje predmet serveru, a server vraća klijentu poruku koji je predmet zamislio, kao i informaciju o tome ko je pobedio

```
Choose object:
1 = Paper
2 = Stone
3 = Scissors

Your selection: 3
You selected Scissors
I select Stone
I won
```

# Ponavljanje zahteva i višekorisnički pristup

---

- Najčešće, serija koraka koja se izvršava na serveru, ponavlja se beskonačno, za jednog ili više klijenata (u našim dosadašnjim primerima to nije bio slučaj)
- Ovaj koncept donosi nova pitanja, kao i nove probleme i njihova rešenja
  - Kako ponoviti već kompletiran proces?
  - Šta ako istovremeno dva ili više klijenata hoće da obave istu operaciju na serveru?
  - Šta ako više klijenata hoće da manipuliše istim resursom na serveru?

I to je samo početak problema



# Ponavljanje zahteva

---

- Svi dosadašnji primeri izvršavaju se linearno i to samo po jednom (biraмо broj, i program se završava, server sabira brojeve i program se završava i slično)
- Petljom možemo omogućiti ponovno konektovanje klijenta
- Ovom prilikom treba biti pažljiv, jer se neke stvari ne isplati ponovno praviti, dok je neke obavezno napraviti ponovo

# Kreiranje servera po potrebi

- U sledećem primeru, ponavlja se kompletan proces kreiranja serverskog soketa
- Ovo je ređi način upotrebe, i dosta je rizičan, jer između dva kreiranja soketa, port može biti zauzet

Mora se zahtevati ponovna upotreba porta

```
import socket
while True:
    sServer = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sServer.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sServer.bind(("localhost", 8005))
    sServer.listen()
    sock, addr = sServer.accept()
    print("Connection received")
    sServer.close()
```

Ponovljen deo

Soket se mora zatvoriti da bi ponovo mogao biti otvoren

# Korišćenje jednog serverskog soketa

- Najčešće se za jedan segment aplikacije koristi jedan serverski soket, dok se sami klijentski soketi generišu iznova za svaku konekciju

```
import socket

sServer = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sServer.bind(("localhost", 8005))
sServer.listen()

while True:
    sock, addr = sServer.accept()
    print("Connection received from " + addr[0])

sServer.close()
```

Samo se jednom kreira  
serverski soket i ne sme se  
uništavati

Ponovljen  
deo

Serverski soket ne sme da bude zatvoren unutar petlje

# Ponavljanje logike nad jednim soketom (pnp-ex01 sum)

- Obično se, serverski soket ne uništava, a klijentski soketi i njihova interna logika, konstantno ponavljaju
- U sledećem primeru jedan korisnik se konektuje na server i unosi brojeve sve dok ne unese prazan string kao vrednost

```
Enter number: 2
Enter number: 4
Enter number: 1
Enter number: 5
Enter number: 3
Enter number: 6
Enter number:
21
```

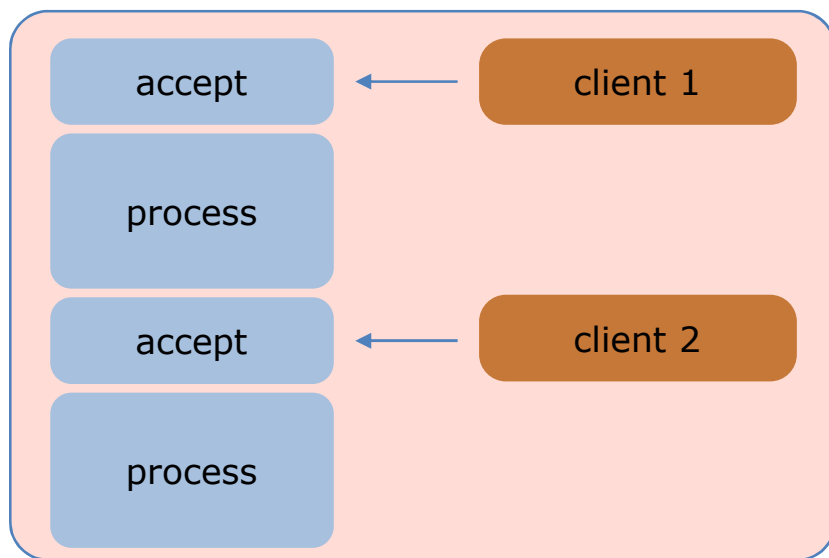
```
import socket
sSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sSocket.bind(("0.0.0.0", 8005))
sSocket.listen()
while True:
    client, addr = sSocket.accept()
    sum = 0
    while True:
        val = client.recv(128).decode("utf-8")
        if val == "#":
            client.send(f"{sum}".encode("utf-8"))
            break
        else:
            sum += int(val)
    client.close()
```

```
import socket
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect(("localhost", 8005))
while True:
    val = input("Enter number: ")
    if val == "#":
        client.send("#".encode("utf-8"))
        print(client.recv(128).decode("utf-8"))
        break
    else:
        client.send(val.encode("utf-8"))
client.close()
```

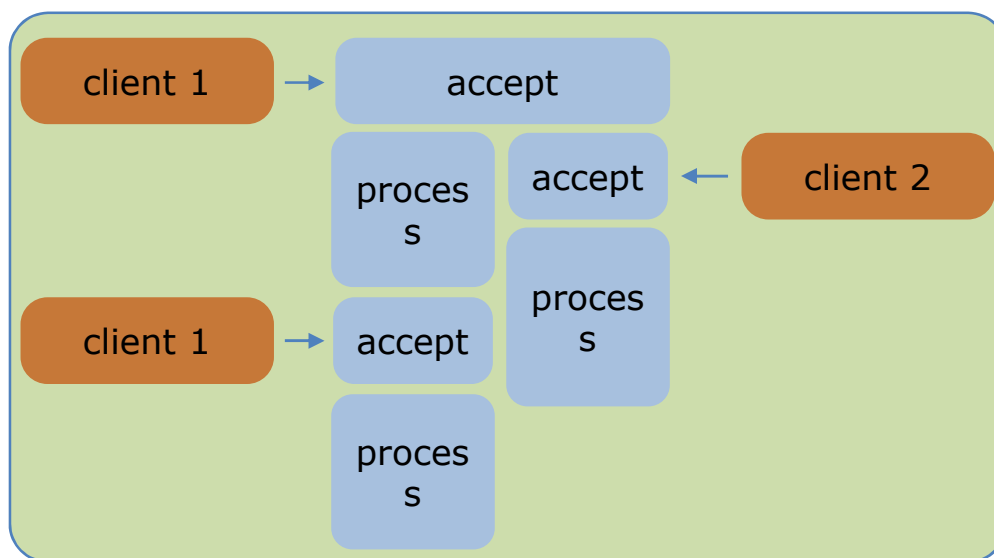
# Rad sa više klijenata

- Problem kod dosadašnjeg pristupa je što server može da obrađuje samo jednog klijenta istovremeno.

Nećemo ovako



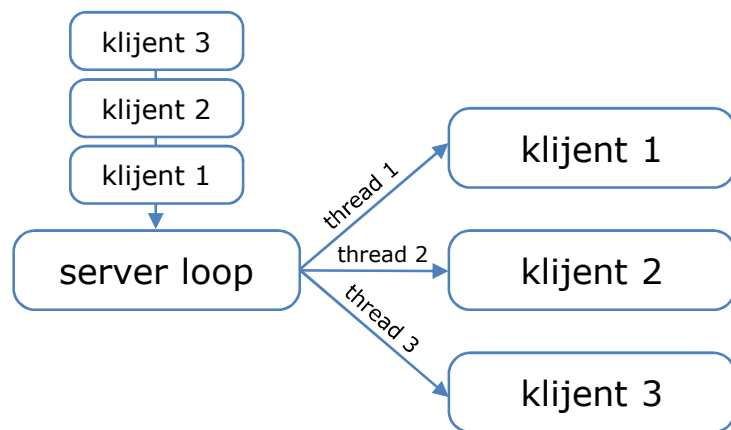
Hoćemo ovako



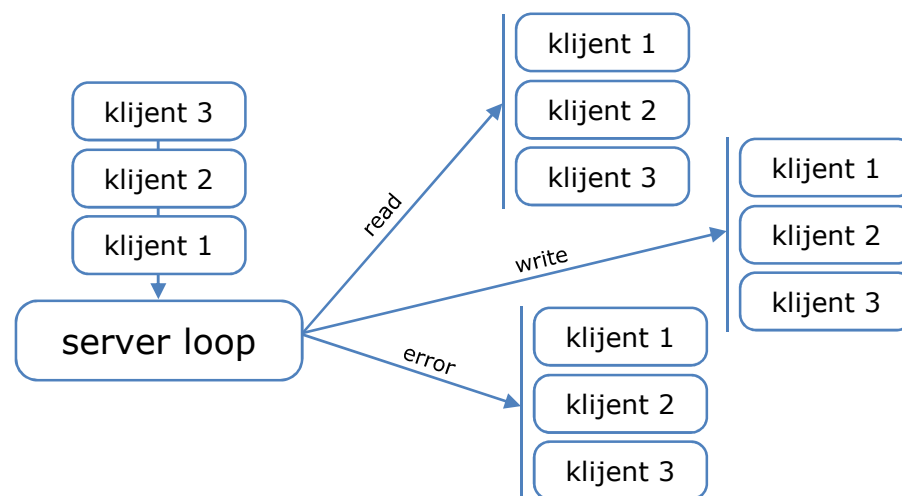
# Rad sa više klijenata

- Procesiranje više klijenata istovremeno, može se rešiti tehnikama **niti** ili **selektora**
- Kod niti, svaki prihvaćeni soket smeštamo u zasebnu nit koja ga zatim procesira. Ova tehnika je jednostavna, ali resursno zahtevna
- Kod selektora, koristimo sistemske događaje nad soketima. Ova tehnika je komplikovanija, ali manje resursno zahtevna

## Thread



## Select





# Select - implementacija (pnp-ex01 helloselect.py)

~~<https://docs.python.org/3.5/library/select.html#module-select>~~

- Funkcija select iz modula select, prihvata tri niza: niz kandidata za detekciju čitanja, upisa i grešaka
- Funkcija select kao rezultat vraća tri niza koji sadrže validne objekte u stanjima za čitanje, upis ili greške
- Sve osim pomenutog, u ovom sistemu moramo uraditi ručno

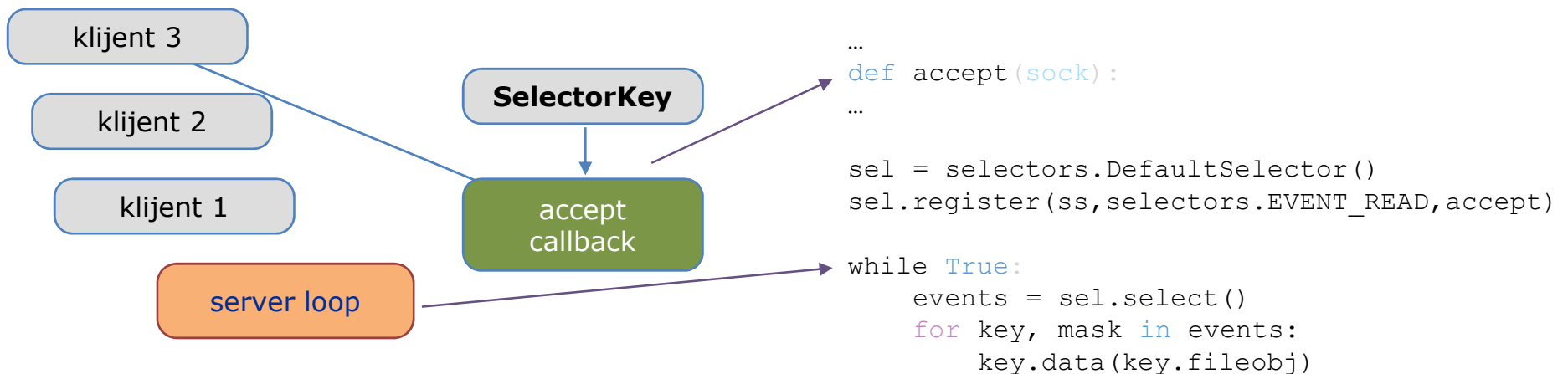
```
while True:
    r,w,e = select.select(inputs,outputs,inputs)
    for sock in r:
        if sock is ss:
            cc,addr = sock.accept()
            inputs.append(cc)
            clients[cc] = []
            outputs.append(cc)
        else:
            total+=1
            sock.recv(1024)
            clients[sock].append(f"HTTP/1.1 200 OK\r\n\r\nTotal {total}")
            print(f"Total {total}",end="\r")
    ...

    for sock in w:
        if sock is not ss:
            if clients[sock]:
                msg = clients[sock].pop(0)
                sock.send(msg.encode("utf-8"))
                outputs.remove(sock)
                inputs.remove(sock)
                del clients[sock]
                sock.close()
```

# Selectors - implementacija (pnp-ex01 helloselectors.py)

~~<https://docs.python.org/3.5/library/selectors.html#module-selectors>~~

- Modul selectors automatski radi ono što se u modulu select mora raditi samostalno pa je njegova upotreba najbolja opcija za rad sa soketima na niskom nivou
- Modul selectors radi po sistemu callback funkcija



# Vežba 4 Chat (pnp-ex01 letstalk)

- Potrebno je napraviti "chat" aplikaciju
- Aplikacija ima klijentski i serverski deo
- Aplikacija podržava više klijenata (ne sme biti chat 1:1)

```
# python3 client.py
Enter username: Peter
Enter message:
User Sally has entered room
Sally: Hello Peter, what's up?
Hmmm...what?
Enter message:
Peter: Hmmm...what?
█
```

```
# python3 server.py
Chat server running...
User Peter has entered room
User Sally has entered room
User Sally say Hello Peter, what's up?
User Peter say Hmmm...what?
█
```

```
# python3 client.py
Enter username: Sally
Enter message:
Hello Peter, what's up?
Enter message:
Sally: Hello Peter, what's up?
Peter: Hmmm...what?
█
```

# Vežba 5 Bingo

---

- Potrebno je napraviti aplikaciju bingo
- U određenim vremenskim intervalima objavljuje se 5 unikatnih brojeva po slučajnom izboru, od 1 do 47
- Korisnici pristupaju serveru, unoseći svoje kombinacije brojeva
- Na kraju svakog izvlačenja, prikazuje se lista izvučenih brojeva, a svakom korisniku se šalje poruka, koliko je brojeva pogodio
- Ukoliko neki korisnik pogodi sve brojeve, dobija poruku BINGO

# AsyncIO TCP soketi

---

- Osim predstavljenih kontrola na niskom nivou, rukovanje soketom je moguće i na višim nivoima

```
import asyncio

async def conn_cbf(r, w):
    await r.read(1024)
    w.write(b"HTTP/1.1 200 OK\r\n\r\nHello world")
    w.close()

loop = asyncio.get_event_loop()
server = asyncio.start_server(conn_cbf, "localhost", 8005)
server_loop = loop.run_until_complete(server)
loop.run_forever()
```

# UDP protokol

---

- UDP protokol - **User Datagram Protocol**

- Ne postoji perzistentna konekcija 

- Paketi su manji, ali je takođe manja i garancija njihove isporuke



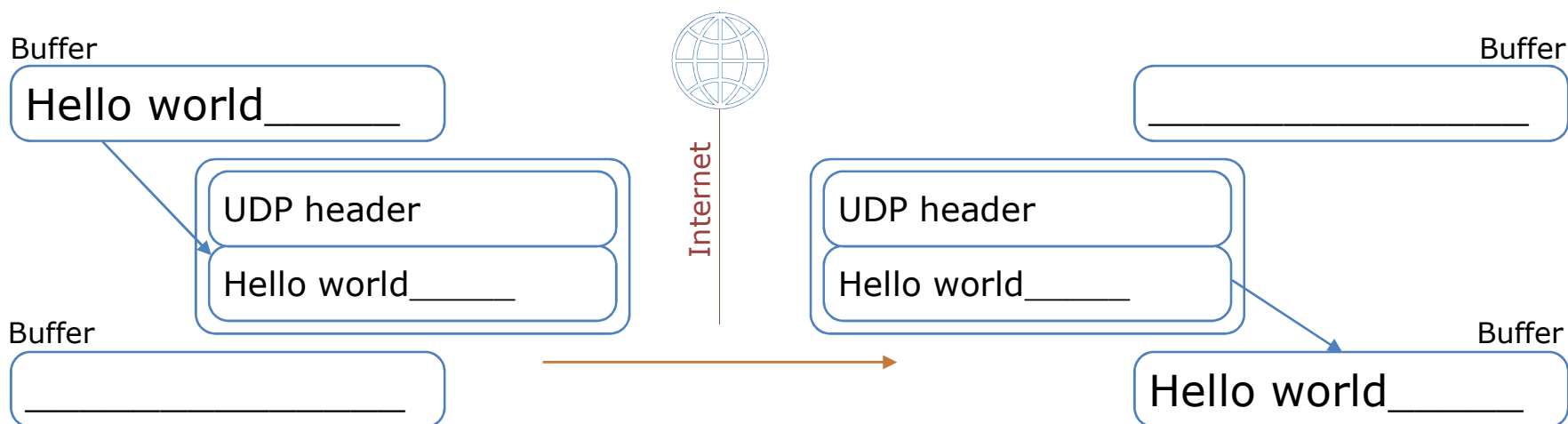
- Paketi ne moraju stići po redosledu slanja



# UDP paketi



- Kod udp protokola, glavni akter je UDP paket. Ovaj paket je vidno manji od TCP paketa, i takođe, njegov sadržaj treba da bude što manji, zbog potencijalnog gubitka
- Prilikom slanja, ili primanja paketa, obično koristimo višekratni bafer bajtova, koji punimo i praznimo za svaki pristigli ili poslati paket



# UDP implementacija ( pnp-ex01 udpserver.py pnp-ex01 udpclient.py )

- UDP soket je i dalje IP socket, pa se za njega može koristiti ista klasa kao i za TCP socket

## UDP server

```
import socket

server = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server.bind(("localhost", 8005))
msg = server.recvfrom(16)
print(msg)
```

Tip soketa nije stream,  
već datagram

Slušanje se obavlja  
metodom recvfrom

(b'Hello world', ('127.0.0.1', 60415))

## UDP klijent

```
import socket

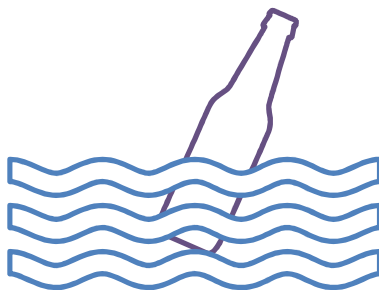
client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
client.connect(("localhost", 8005))
client.send(b"Hello world")
```



# Problemi UDP-a

---

- UDP nije strimovana konekcija
- Ne postoji mogućnost sinhronizacije između pitanja i odgovora (tačnije, nije ni moguće dati odgovor)
- UDP poruka je osuđena na milost i nemilost mreže, poput poruke u boci



- Jedan od najvećih problema UDP-a jeste nemogućnost lociranja sagovornika

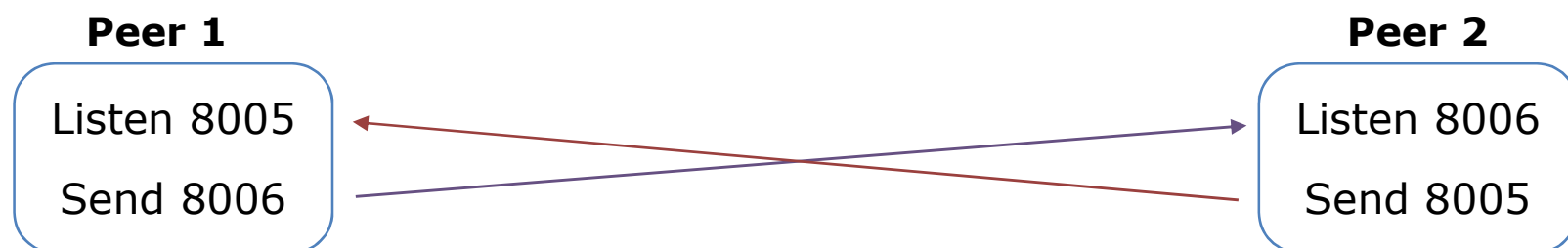
# UDP razmena poruka između sagovornika

---

- Rešenje za razmenu poruka između sagovornika u UDP-u jeste da oba sagovornika budu i klijent i server
- Problem je što serverske osobine zahtevaju zauzimanje porta (bind), a slušanje na istom portu, na istom operativnom sistemu, nije moguće
- Implementacija ovog koncepta je, zbog toga, za ovaj scenario, moguća na jedan od dva (ili oba) načina:
  - Korišćenje različitih portova između sagovornika
  - Slanje poruke i momentalno aktiviranje prijemnog soketa kod prvog, odnosno prijem poruke i momentalno gašenje prijemnog soketa kod drugog sagovornika
- Može se iskoristiti još uvek otvoreni port za momentalni odgovor

## USP dva klijenta i servera (pnp-ex01 udpclientserver)

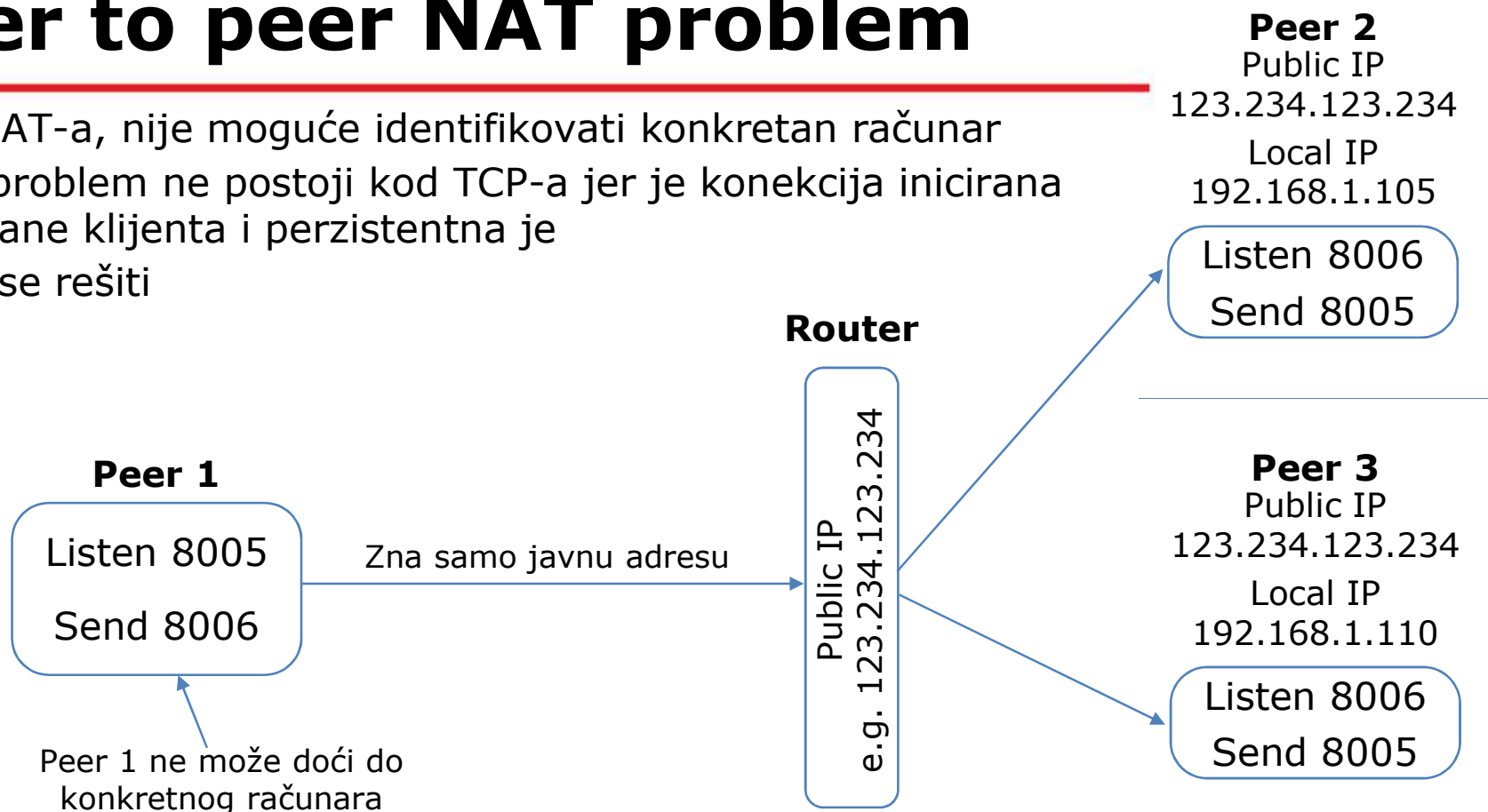
- U ovom scenariju, oba sagovornika imaju ravnopravnu infrastrukturu. Na jednom portu slušaju, a na drugom emituju
- Ovakav koncept zove se **peer to peer**, jer oba sagovornika imaju istu važnost u komunikaciji



- Problem je što peer to peer ne može (podrazumevano) funkcionisati u slučaju NAT-a (Network Address Translation)

# Peer to peer NAT problem

- Kod NAT-a, nije moguće identifikovati konkretan računar
- Ovaj problem ne postoji kod TCP-a jer je konekcija inicirana od strane klijenta i perzistentna je
- Može se rešiti



# Peer to peer NAT rešenja (pnp-ex01 udpopenport)

- Jedno rešenje je korišćenje trećeg računara preko koga bi dva peer-a komunicirala ili samo izvršila "otkrivanje".
- Drugo rešenje je korišćenje još uvek otvorenog porta kako bi se poslao "odgovor" pošiljaocu udp paketa (ova funkcionalnost zavisi i od mrežne infrastrukture, jer ruter mora da pretpostavi da se očekuje UDP paket ka lokaciji sa koje je poslat, a sa lokacije na koju je poslat)

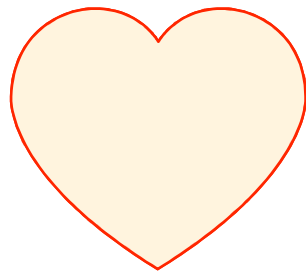
```
server = socket.socket(...)
server.bind(("localhost", 8005))
data = server.recvfrom(128)
print(data[0].decode("utf-8"))
server.connect(data[1])
server.send(b"Hello from me also")
data = server.recvfrom(128)
print(data[0].decode("utf-8"))
server.send(b"Woow..we speak on the edge")
```

```
client = socket.socket(...)
client.connect(("localhost", 8005))
client.send(b"Hello")
print(client.recv(128).decode("utf-8"))
client.send(b"Hello again")
print(client.recv(128).decode("utf-8"))
```

# Održavanje UDP veze (pnp-ex01 udpheartbeat)

---

- Pošto konekcija u udp-u nije perzistentna, može biti prekinuta (timeout)
- Da bismo osigurali vezu, sagovornici konstantno šalju poruke (makar i bez smisla) kako bi jedan drugome dali do znanja da su i dalje prisutni
- Ovaj koncept se zove **keep alive**, a poruke **heartbeats**.



# Multicast (pnp-ex01 ...)

- Multicast je poseban režim rada karakterističan za UDP
- U ovom režimu, definiše se grupa (multicast grupa) čiji pripadnici će dobiti poslatu poruku
- Multicast koristimo kada ne postoji individualan sadržaj, već svi korisnici dobijaju isti sadržaj (strimovanje, notifikacija i slično)

```
multicast_group = ('224.0.0.1', 1001)
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.setsockopt(socket.IPPROTO_IP, socket.IP_MULTICAST_TTL, 1)
sent = sock.sendto(b'Hey all!', multicast_group)
```

```
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind(('', 1001))
group = socket.inet_aton('224.0.0.1')
mreq = struct.pack('4sL', group, socket.INADDR_ANY)
sock.setsockopt(socket.IPPROTO_IP, socket.IP_ADD_MEMBERSHIP, mreq)
data, address = sock.recvfrom(1024)
print(data.decode("utf-8"))
```

